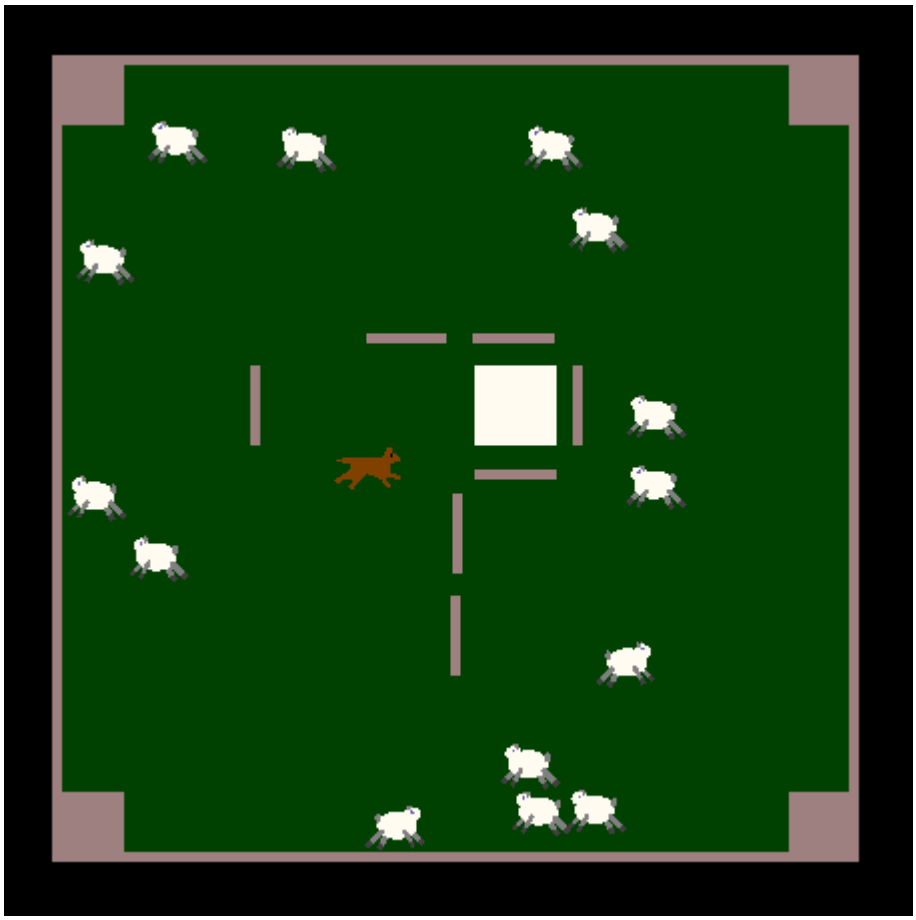
Sheepdog

User Manual ■ Program Design Notes ■ Future Recommendations



Jennifer Beers
CSS 450, 2D Graphics
University of Washington
December 2001

Table of Contents

Table of Contents 1
Introduction 1
User Manual 1
- Using the Detail View 2
- Using the Map to Zoom and Pan 2
- Step 1: Designing a Landscape 2
  - Tips for Designing Landscapes 2
- Step 2: Creating a Flock of Sheep 3
  - Tips for Creating a Flock 3
- Step 3: Adding a Sheepdog 3
  - Sheepdog Characteristics 3
  - Customizing Sheepdog Appearance 4
  - Tips for Creating a Sheepdog 4
- Step 4: Playing the Game 4
  - Moving the Sheepdog 4
  - Pause, Resume, and Reset 4
  - Game Status 5
  - Tips for Playing the Game 5

Design Notes 5
- The Graphical User Interface 5
- The Graphic Object Class Hierarchy 7
- The Game Logic Class 7

Recommendations for the Future 9
- Saving and Loading 9
- Improved Object Bounds 9
- Improved Graphic Object Hierarchy 9
- Additional Customization for Level Design 9
- Better Graphics, Animation and Sound 9
- More Realistic Animal Movement 9
- Requiring Playable Levels 10
- Fix Known Problems 10
- General Code Improvements 10

Introduction

Sheepdog is an interactive computer game with 2-dimensional computer graphics. Users can create a game level by designing a landscape and building a customized flock of sheep. They can then play the level that they have created. To win the game, the user must herd each of the sheep into a sheep pen, before the sheepdog becomes exhausted.

This documents contains the user manual for the game, and a description of the program design. In addition, there are recommendations for future features and design improvements. The document assumes that the reader is familiar with 2-dimensional computer graphics and event-driven programming.

User Manual

The Sheepdog program begins in level design mode. You go through three steps to design a level, and then play the game. The objective is to herd every sheep into a sheep pen.

In both level design mode and the game play mode, the main game window shows two views of the sheep pasture (see Figure 1). Graphical User Interface (GUI) controls show on the left, a *Detail View* of the sheep pasture shows in the middle, and a smaller *Map* of the pasture shows on the right. During level design (Step 1 through Step 3), you will see a message box at the bottom of the GUI controls. This box

will show messages about how to design your game level. Figures 1 through 3 show how the game appears during the three level design steps and actual game play.

Using the Detail View

When you are designing levels during Steps 1 through 3, you can left click in the *Detail View* to add objects. After you click down, you can drag the mouse to move the object. When you release the left mouse button, the object is added to the level.

Using the Map to Zoom and Pan

The *Map* shows the entire pasture area. A white box shows which part of the pasture is displaying in the *Detail View*. You can use the mouse on the *Map* to change what appears in the *Detail View*. To zoom, make a middle-button click in the M*ap* and drag the mouse. Drag left to zoom in, or drag right to zoom out. To pan, right click in the map view and drag. The white box moves as you move the mouse, changing which part of the pasture displays in the *Detail View*. Note: You can zoom at any time. However, once you start playing the game, you can only pan when the game is paused. When the game is running, the Detail View is centers on the sheepdog.

Step 1: Designing a Landscape

You can design your own landscape by adding obstacles and designated S*heep Pen* areas. See Figure 1. Choose one of the three buttons in the GUI controls: Sheep Pen Area, Horizontal Fence, or Vertical Fence. Then left click in the *Detail View*. After you have entered at least one sheep pen you can move to Step 2. When you are happy with your landscape, press the [Next: Add Sheep] button.

Tips for Designing Landscapes

- The more sheep pens you have, the easier it will be to play the game.
- The more fences you have, the more difficult it will be to play.
- Do not surround pens with fences that completely block sheep from getting in.

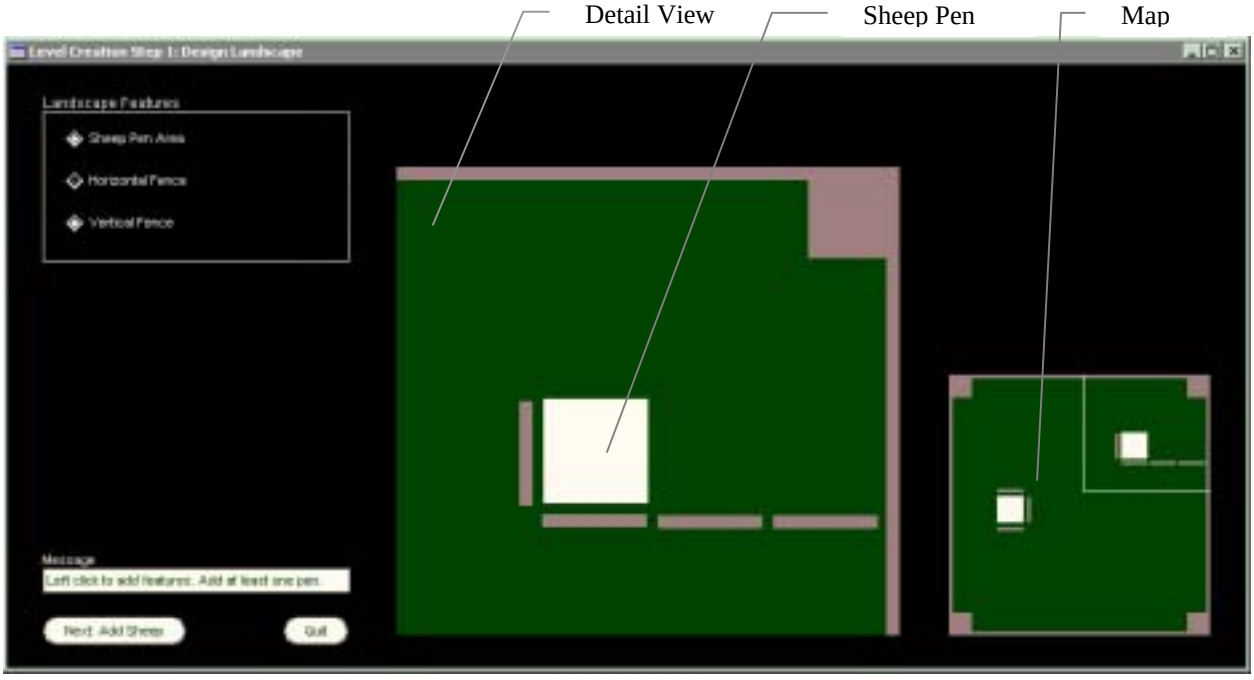


Figure 1: Level Creation Step 1: Design Landscape

Step 2: Creating a Flock of Sheep

You can design your own flock of sheep. See Figure 2. Set the *Speed* and *Reaction Distance* sliders to the settings you want for a sheep. Then left click in the *Detail View* to add a sheep. The sheep you add will have the settings that show on the *Speed* and *Reaction Distance* sliders. You can also adjust settings with these sliders immediately after a sheep has been added. Each sheep can have its own independent settings. After you have entered at least one sheep you can move to Step 3. When you are happy with your flock settings, press the [Next: Add Sheepdog] button.

Tips for Creating a Flock

- The more sheep you have, the harder the game will be.
- Faster sheep are harder to herd than slower sheep.
- Sheep are easiest to herd when they react to a sheepdog within 3 or 4 sheep widths away. If they react to dogs at a farther distance, it can be very difficult for the dog to herd them away from the edges of the pasture. If they only react to dogs within 2 sheep widths away, they are easy to herd, but it takes the sheepdog more leaps to get close enough to affect the sheep.
- Make sure to leave enough room to add a sheepdog.
- Make sure that you do not put a sheep in position where landscape features prevent it from ever reaching a pen.

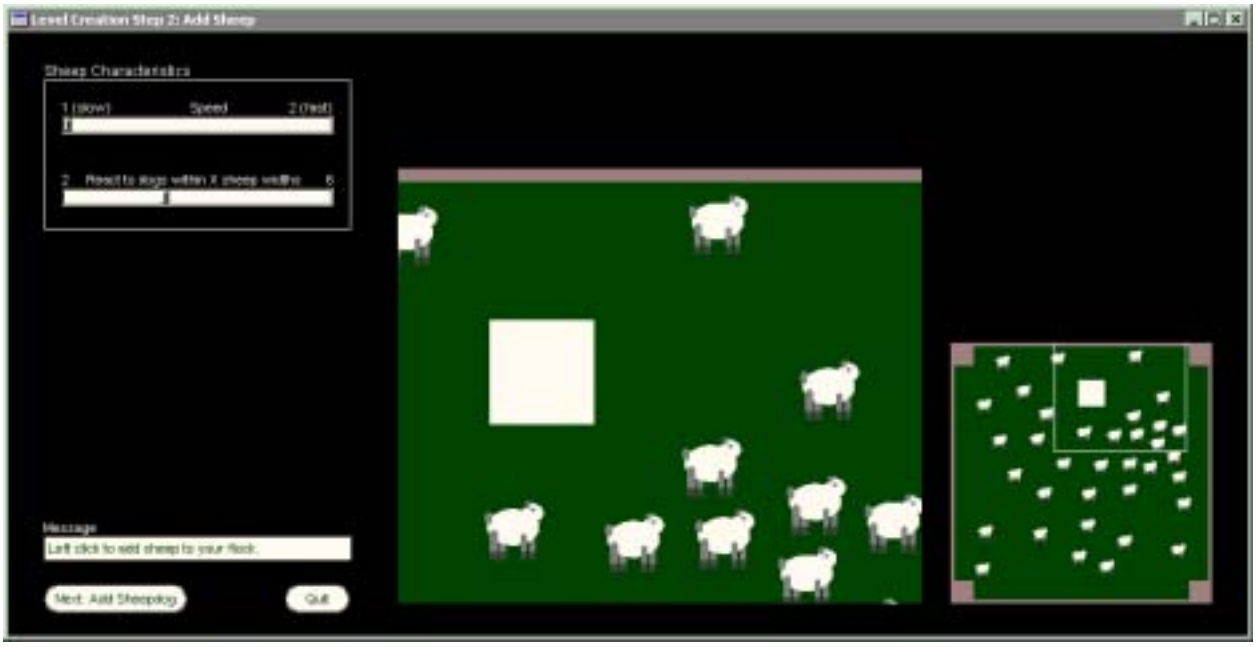


Figure 2: Level Creation Step 2: Add Sheep

Step 3: Adding a Sheepdog

You can only have one sheepdog, but you can customize many of its aspects. See Figure 3. Decide where you want the sheepdog to be, and then left click in the *Detail View*. You can then adjust the sheepdog settings, which are described below. When you are happy with your sheepdog settings and ready to play, press the [Next:Play!] button.

Sheepdog Characteristics

The top panel in the GUI controls contains Sheepdog Characteristics. You can use the first two sliders to rotate the dog and change its size. The third slider sets the sheepdog's endurance between 10 to 500 leaps. To win the game, you must herd all the sheep into pens before the dogs runs out of leaps.

Customizing Sheepdog Appearance

The bottom panel in the GUI controls contains Limb settings. You can select a limb button, and then change the rotation, position, and size for that limb. Adjust the limbs so you're you like the appearance of the leaping position.

Tips for Creating a Sheepdog

- Make sure that you do not put the sheepdog in position where landscape features prevent it from reaching the sheep.
- The larger the sheepdog, the harder it is to herd sheep.
- If you rotate the dog at an angle (instead of straight up, down, or sideways), it will take up more space. This will make it harder to herd sheep.
- The lower the sheepdogs endurance, the harder it is to win the game.
- Do not size or rotate the dog so that it overlaps with landscape features or sheep. If you do, the dog will not be able to move.

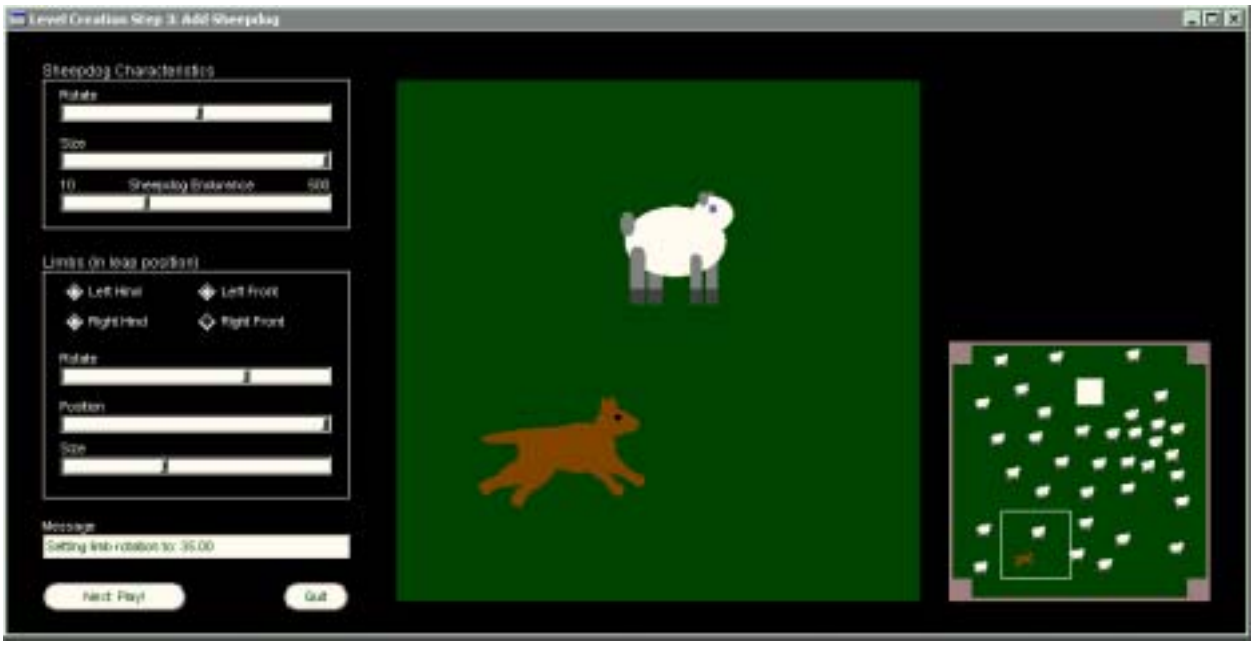


Figure 3: Level Creation Step 3: Add Sheepdog

Step 4: Playing the Game

Once you begin playing, you must act quickly to herd each of the sheep into a sheep pen. As soon as a sheep touches any part of the pen, it will leap into the middle of the pen and disappear. If you herd all the sheep before you run out of leaps, you win! If you run out of leaps before you herd all the sheep, your sheepdog becomes exhausted, and the game is over. When the game ends, whether you win or lose, the game will stop, and you can see your results on the GUI controls, as described below.

Moving the Sheepdog

You can move the sheepdog in four directions: right, left, up, and down. You can click on the direction buttons in the GUI control, use the arrow keys, or use the following letter keys: a = left, d = right, w = up, x = down.

Pause, Resume, and Reset

Choose [Pause] at any time to freeze the game. Choose [Resume] to restart the game. When the game is paused, you can pan using the *Map* (see Using the Map to Zoom and Pan). When the game resumes, you can no longer pan. The Reset feature is not yet implemented.

Game Status

While you are playing the game, the GUI controls show the game status. You can see how many sheep you have successfully herded into sheep pen areas, and how many you have left to herd. You can also see how many leaps you have left. You can pan and zoom using the *Map*, but you cannot do anything else. Press [Quit] to end the game.

Tips for Playing the Game

- Once a sheep is heading toward a pen, you can go herd other sheep. As long as you do scare the sheep in another direction, and it does not hit an obstacle or another sheep, it will continue in to the pen on its own.
- If a sheep is moving along the pasture wall, stay away from it until it gets to a corner and starts moving a little bit away from the next wall. You can then run up behind it and herd it away from the wall. It is easier to do this on the top and bottom walls than the side walls, because your dog is shorter than it is wide.

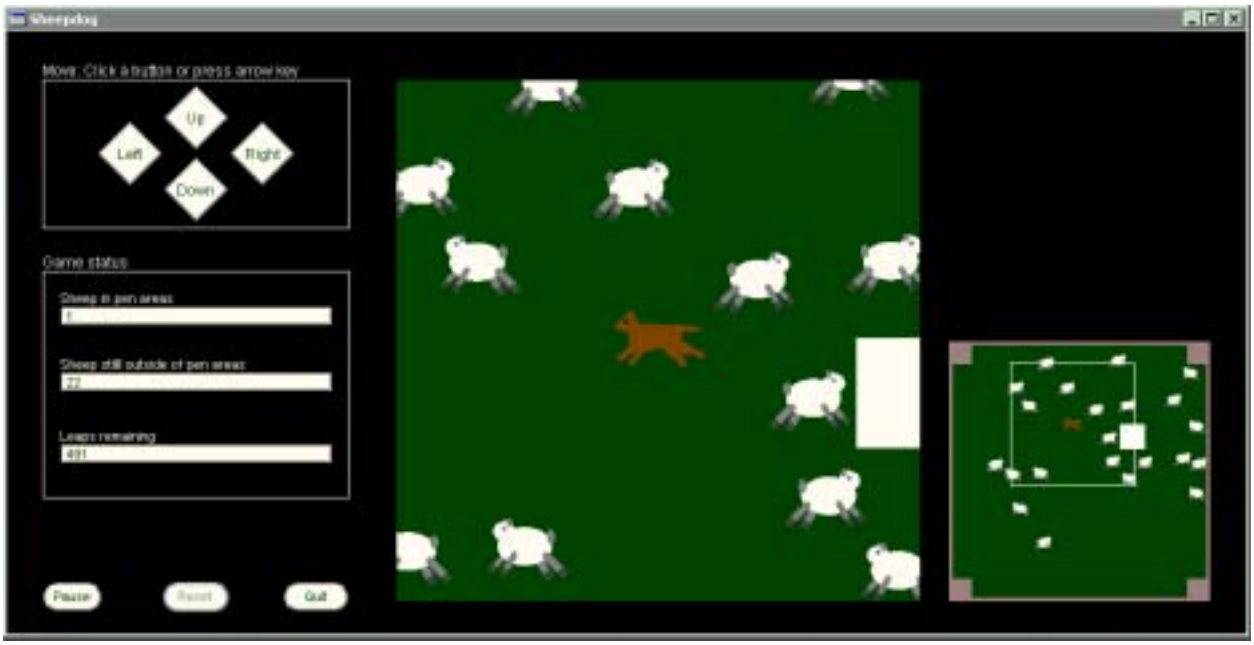


Figure 4: Playing the Game

Design Notes

The sheepdog game contains three main components: the graphical user interface (GUI), the graphic object class hierarchy, and the game logic class. The section of the document assumes that you have read the user manual and/or played the sheepdog game.

The Graphical User Interface

The sheepdog game user FLTK and GLUT to manage user input and output. There is one GUI class, SD_Control, used for all stages of the game. As the user moves through the level design steps, the game logic class, SD_Game, determines which GUI controls are showing and which are hidden. SD_Game::nextStep() contains most of the code that alters the appearance of the GUI (beyond just control values). The game steps work as shown in Figure 5.

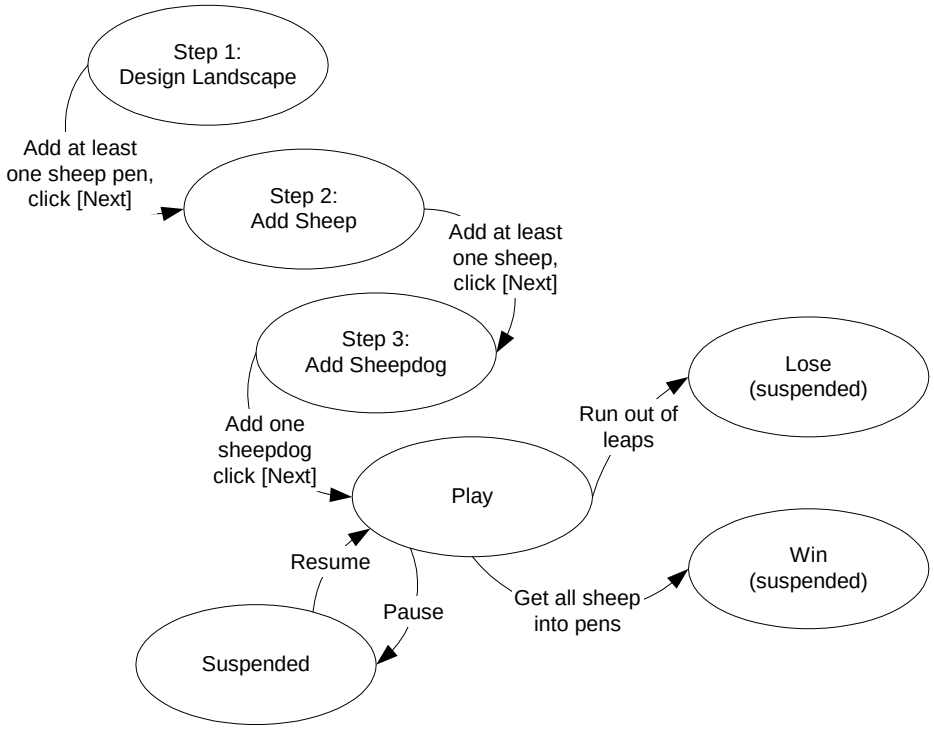


Figure 5: Main game states

In some cases GUI controls use callback methods in the ServiceGUI.cpp file to interact with SD_Game. In other cases, SD_Game methods get values directly from the GUI controls as needed. The ServiceEvents file contains callbacks for mouse events and keyboard actions, to supports the features shown in Tables 1a, 1b, and 1c. When the game is suspended, it takes no actions in response to mouse or keyboard events. In the Lose or Win state, the game is suspended. When the game is being played, the editView centers on the sheepdog. Panning is prevented during game play, except when the game is paused.

Mouse Actions During Level Design: Steps 1 through 3		
Mouse Event	**Action in editView GraphicsState** (larger center viewport)	**Action in mapView GraphicsState** (smaller viewport on the right)
Left/Down	Add the current object type (landscape feature, sheep, or sheepdog) at the position clicked.	-
Left/Drag	Drag the object which was just added with Left/Down.	-
Left/Up	Insert the object into the appropriate game collection, if there is no collision.	
Middle/Down	-	Records the mouse position
Middle/Drag	-	Left motion zooms into the game world, right motion zooms out of the game world. Zoom amount is determined from the starting click position.
Right/Down	-	Moves the origin of the *Zoom Box* (and *Play* view), to the clicked position.
Right/Drag	-	Drags the *Zoom Box*.

Table 1a: Mouse events and actions during level design

Mouse Actions During Game Play		
Mouse Event	**Action in editView GraphicsState** (larger center viewport)	**Action in mapView GraphicsState** (smaller viewport on the right)
Middle/Down	-	Same as Level Design (above)
Middle/Drag	-	Same as Level Design (above)
Right/Down	-	If game is suspended, same as level design above. Else, does nothing.
Right/Drag	-	If game is suspended, same as level design above. Else, does nothing.

Table 1b: Mouse events and actions during game play

Keyboard Actions During Game Play	
Key -> or 'a'	Change sheepdog direction to down
Key -> or 'd'	Change sheepdog direction to down
Key -> or 'w'	Change sheepdog direction to down
Key -> or 'x'	Change sheepdog direction to down

Table 1c: Keyboard events and actions during game play

The Graphic Object Class Hierarchy

All non-GUI images shown in the game are built with the GO_GraphicsObj hierarchy, which uses OpenGL. See Figure 6. The game contains the graphic objects shown in Table 2.

1	GO_GraphicObj*	currentGO	Used during level design
2	GO_GraphicObj*	GOToBeAdded	Used during level design
3	GO_Rectangle*	currentBoundBox	Displays collision boundaries during level design
4	GO_GraphicObj*	pasture	The green background for the game
5	GO_Collection*	pastureFences	Initial landscape obstacles
6	GO_Collection*	sheepPens	Sheep pen areas
7	GO_Collection*	landscape	Landscape obstacles added by the user
8	GO_Collection*	sheep	Sheep added by the user
9	GO_GraphicObj*	sheepdog	The sheepdog, added by the user
10	GO_Collection	sceneGOs	A collection that points to GOs 4 - 9

Table 2: GO_GraphicObj objects in the Sheepdog game.

The GO_GraphicObj::getObjectBound method supports collision detection. Currently, an object bound is the smallest rectangle that completely contains an object. This method returns a known invalid bound when use for a GO_Collection object.

The Game Logic Class

Once the game is in play, the *Sheepdog* responds to user-initiated actions that set direction, and to timer events. The *Sheep* respond to timer events. Neither the *Sheep* nor the *Sheepdog* can move into positions where they will collide with other animals or landscape obstacles.

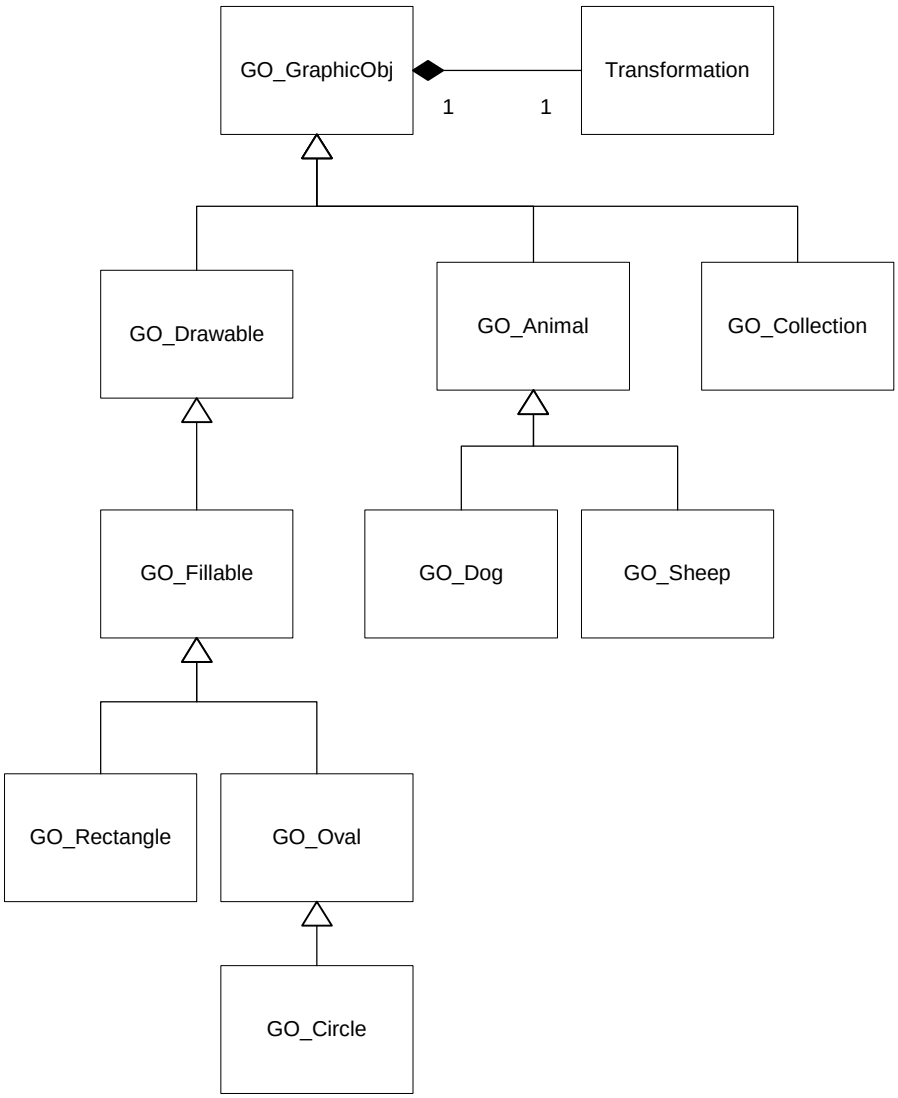


Figure 6: The GO_GraphicsObj class hierarchy

When the user changes the sheepdog direction, the direction is stored as a state in the GO_Dog object. However, no visible action occurs until the periodic timer event occurs. At this time, the entire game state is updated. The general description of the SD_Game::updateGameState() process is as follows.

- Iterate through the sheep collection, and delete any sheep that have moved into pens during the last timer event.
- Iterate through remaining sheep.
    - If the dog is nearby, change direction to face away from it.
    - Try to move. If movement results in a collision or faces toward a nearby dog, turn 90 degrees and try another direction (this creates better herd behavior than turning 180 degrees). Stop if none of the four directions are clear.
    - Check if the sheep is in a pen. If so, move it to the center of the pen, set its transformation to show a rotated, angled leap, and mark the sheep for deletion.

        - Change the sheep from standing to leaping (or vice versa), even if it did not move to a new location.
    - Try to move the sheepdog in its current direction. If there is a collision, stand still. (The sheepdog does not turn away from objects, because that would seriously detract from its ability to herd sheep). Change the sheepdog from standing to leaping (or vice versa), even if it did not move to a new location.
    - Scroll the editView so that centers on the sheepdog.
    - Update game statistics.
    - Check if the game has been won or lost. If so, update the GUI and suspend the game.

    Because n sheep check for collisions with n sheep and m landscape features, the algorithm is O(n[n+m]). It slows the game considerably when there are a large number of sheep. The game difficulty remains consistent because the dog speed is still relative to the sheep speed, but a faster algorithm would be better. See this and many other notes in the Recommendations for the Future section below.

Recommendations for the Future

Many additional features and code improvements could be added to the sheepdog game.

Saving and Loading

In the future, level designers should be able to save their work as a level, and then reload a single level or a series of levels. Level designers could also provide a difficulty rating for their level, to be used for more complex scoring. Alternately, the Resume feature could be implemented by doing a deep copy of the sheep collection and copying the sheepdog. When the player clicked the resume button, the existing sheep and sheepdog would be deleted, and replaced with copies of the originals.

Improved Object Bounds

The graphics object class should support bounds that fit tightly around the subcomponents of an object. This would support more accurate collision detection, and include rotated limbs in boundary calculations.

Improved Graphic Object Hierarchy

GO_GraphicsObj contains several methods that really only apply to GO_Animal. This hierarchy should be redesigned to support movement and animation more efficiently.

Additional Customization for Level Design

The level design steps could be enhanced to become a more fully-featured graphic editing environment. The user could create custom obstacles, such as ponds, rocks, and animated gates that open and close after a specified series of timer events. In addition, there are many additional appearance and behavior attributes that could be added for sheep and sheepdogs.

Better Graphics, Animation and Sound

Texture maps, sounds and more complex more animals shapes and animation sequences would all enhance this game. Specific events worth highlighting include: the sheep first reacting to a nearby dog, the sheep entering a pen, the dog becoming exhausted (losing), and the dog herding the entire flock (winning).

More Realistic Animal Movement

The sheep could use a more sophisticated algorithm to prevent “wall crawling”, rather than requiring the redirector landscape obstacles in the corner of the pasture. It would also be great if the sheep changed speeds and followed other sheep that passed nearby (herding instinct).

Currently, the sheep can appear to panic (rapidly and repeatedly switch between two directions) when they are partially cornered by a dog. This is a neat effect for a simple game, and it doesn't block game play. However, the direction code should be fixed so that the sheep can always find a direction if one is available. It would also be helpful to allow the user to have the sheepdog halt whenever the user wants, without requiring it to come up against an obstacle.

Requiring Playable Levels

The game currently encourages users to make playable games, but it would be great to add logic that prevent some of the remaining unplayable levels, such as having one sheep pen, sheep, or sheepdog completely blocked by obstacles.

Fix Known Problems

A few known problems should be fixed.

- Users can place objects outside of the pasture.
- If a user left clicks to insert an object, drags outside the editView, and then releases, the object still displays. It is not added to sceneGO's, and disappears when the user left clicks somewhere in editView.
- If the user left clicks outside the editView, then drags into the editView and releases, the previously added object gets moved to the mouse up location. This can result in an invalid state where the game does not have a sheep or sheepdog. The left click and drag code needs to be revisited, and possibly redesigned.
- The sheepdog can be rotated so that it collides with an obstacle or sheep. In this case, the sheepdog can never move. Collision detection could be added to prevent this case.
- If a pen is partially blocked by fences, but a small corner sticks out, the sheep can touch the corner and magically squeeze through the small opening into the pen. This could be fixed by putting in collision detection along an entire path of movement.
- When the game is lost, the sheepdog rotates 180 degrees. When it does this, it can rotate to collide with sheep and fences. It would be better to vertically center the origin of GO_Dog, then flip the dog.

General Code Improvements

In general, the sheepdog code was written with consistent coding conventions. However, the last 10 percent of code was added quickly, and should be cleaned up. For example:

- There are a handful of some unnamed literals that should be pulled out and put into appropriate class headers.
- Some methods names are legacies of the game initial design and are no longer useful for the current game.
- Some of the SD_Game functions should be made private instead of public.
- The UI implementation could be improved. Perhaps the game could bring up a different window for game play, rather than showing and hiding controls. Also, controls such as [Next] should be visible, but greyed out, rather than hidden.
- Some code in the Game API library is not used in the game and could be removed from the executable using IFDEF statements.